Joint Inventors

Docket No. Intel/P16810

"EXPRESS MAIL" mailing label No.
EV 309992037 US
Date of Deposit: **August 26, 2003**

I hereby certify that this paper (or fee) is being deposited with the United States Postal Service "EXPRESS MAIL POST OFFICE TO ADDRESSEE" service under 37 CFR §1.10 on the date indicated above and is addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450

Magda Greer

Magda Greer

# APPLICATION FOR UNITED STATES LETTERS PATENT

# S P E C I F I C A T I O N

TO ALL WHOM IT MAY CONCERN:

Be it known that We, Ping T. TANG, a citizen of United States of America, residing at 716 City Walk Place #2, Hayward, California 94541; and Gopi K. KOLLI, a citizen of India, residing at 17 Concord Greene Apt.4, Concord, Massachusetts 01742; and Minda ZHANG, a citizen of United States of America, residing at 3 Patten Lane, Westford, Massachusetts 01885 have invented new and useful **METHODS AND APPARATUS FOR DETERMINING APPROXIMATING POLYNOMIALS USING INSTRUCTION-EMBEDED COEFFICIENTS**, of which the following is a specification.

# METHODS AND APPARATUS FOR DETERMINING APPROXIMATING POLYNOMIALS USING INSTRUCTION-EMBEDDED COEFFICIENTS

## FIELD OF THE DISCLOSURE

**[0001]** The present disclosure relates generally to processor systems and, more particularly, to methods and apparatus for determining approximating polynomials using instruction-embedded coefficients within processor systems.

## BACKGROUND

**[0002]** Algebraic and transcendental functions are fundamental in many fields of application. In particular, *K*-th root family functions of the form $(y)^{\pm\frac{1}{K}}$, which include inverse functions, inverse square root functions and square root functions, are performance critical in many graphics applications. Traditional algorithms for these *K*-th root family functions are typically tailored for desktop computers (e.g., personal computers) and workstation platforms. These traditional algorithms typically provide relatively high precision and accuracy, ranging from approximately seven significant decimals (e.g., IEEE single precision floating point) to sixteen significant decimals (e.g., IEEE double precision floating point). Due to typical accuracy requirements, methods for calculating *K*-th root family functions usually require data memory accesses, which may require the computers or platforms on which the methods are implemented to have relatively large main memories and data caches.

**[0003]** Many emerging classes of handheld computing platforms such as, for example, handheld platforms based on the Intel® XScale™ processor family, rely heavily on *K*-th root family function approximation values. In particular, computer graphics capabilities and performance are highly dependent on the performance of the platform responsible for determining *K*-th root family function approximation values.

However, when traditional *K*-th root family function computational methods are implemented on emerging classes of handheld platforms, these traditional computational methods often result in low and unpredictable performance because data memory accesses often affect the data memory access performance (e.g., corrupt the data cache) of a running application that calls the *K*-th root family functions.

**[0004]** The data memory access required by traditional methods for determining *K*-th root family function approximation values is due in part to the fact that these methods generally require function values to be calculated prior to a compilation phase and stored in a table in data memory. In addition, these traditional methods usually employ general polynomials having coefficients that are stored in data memory during a compilation phase.

**[0005]** Alternative methods for determining *K*-th root family function approximation values that do not require a table of pre-calculated function values have recently been developed. However, these alternative methods typically rely on polynomial functions that include coefficients that are not stored explicitly. Although these alternative methods have provided some improvement over the methods that use pre-calculated function values and tables stored in data memory, the polynomials used by these methods are restrictive and the accuracy of the final result (i.e. the *K*-th root family function value) is relatively low.

**[0006]** Another method for determining *K*-th root family function approximation values uses floating-point arithmetic. However, the use of floating-point arithmetic requires software emulation, which may decrease the overall performance of a processor based-platform when processing *K*-th root family functions.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[0007]** Fig. 1 is a flow diagram illustrating an example method for determining and storing approximating polynomial coefficient values.

**[0008]** Fig. 2 is a flow diagram illustrating another example method for determining approximating polynomial coefficient values.

**[0009]** Fig. 3 is a flow diagram illustrating an example method for determining a runtime approximating polynomial value of an inverse function using instruction-embedded polynomial coefficient values.

**[0010]** Fig. 4 is a flow diagram illustrating an example method for determining a runtime approximating polynomial value of an inverse square root function and a square root function using instruction-embedded polynomial coefficient values.

**[0011]** Fig. 5 is a flow diagram that depicts an example method for performing a self-correcting process that may be used to determine a function approximation value based on an intermediate function approximation value.

**[0012]** Fig. 6 is a block diagram of an example processor system that may be used to implement the apparatus and methods described herein.

## DETAILED DESCRIPTION

**[0013]** The disclosed methods, apparatus and articles of manufacture may be used to calculate a runtime polynomial associated with a runtime approximating polynomial function of any transcendental or algebraic function. In particular, determining a runtime approximating polynomial function is described herein in connection with a $K$-th root family function of the form $(y)^{\pm\frac{1}{K}}$, where $K$ is an exponent scaling value and may be equal to any relatively small positive integer value (i.e., 1, 2, 3, etc.). The disclosed methods, apparatus and articles of manufacture may

be used during a runtime phase within a processor system and may be carried out using only instruction memory accesses (i.e., without requiring data memory accesses). In particular, the examples described herein determine a runtime approximating polynomial by using approximating polynomial coefficient values that are stored in processor instructions during a compilation phase.

**[0014]** Processors such as, for example, processors from the Intel® XScale™ processor family, are capable of processing instructions that include stored coefficients. With these types of processors, an instruction may include an opcode bitfield associated with an executable operation and at least one bitfield associated with a coefficient value. The coefficient value may be used by the processor to execute an operation according to the opcode bitfield. In the case of an Intel® XScale™ processor, an 8-bit coefficient value may be stored within the coefficient bitfield of each instruction. However, the methods, apparatus and articles of manufacture described herein are not limited to processors capable of having only 8-bit coefficient values stored in an instruction, nor are they limited to use with processors from the Intel® XScale™ processor family. To the contrary, the methods, apparatus and articles of manufacture described herein may be used with any processor that supports the use of coefficient values within instructions.

**[0015]** As described in connection with the examples herein, approximating polynomial coefficients may be determined prior to a compilation phase so that during the compilation phase the approximating polynomial coefficients are embedded or otherwise stored in an instruction. For example, a coefficient value 166 may be stored in a multiplication instruction using the following program language.

MUL U, X, #166

**[0016]** During a compilation phase, a compiler may compile the above program language and store the coefficient value 166 in a bitfield associated with the multiplication instruction. Additionally, the coefficient value 166 and its associated multiplication instruction may be stored in an instruction memory of a processor system and may be used during a runtime phase. Two example methods for determining approximating polynomial coefficients are described in greater detail in connection with Figs. 1 and 2. However, other example methods for determining the approximating polynomial coefficients may be used instead.

**[0017]** In addition, the approximating polynomials determined in Figs. 1 and 2 include third-degree polynomials. However, as shown in Equation 1, a polynomial of any degree may be used to approximate any transcendental or algebraic function (e.g., the $K$-th root family function $(y)^{\pm\frac{1}{K}}$).

Equation 1 $$p_A(x) = p_0 - p_1 \cdot x + p_2 \cdot x^2 - p_3 \cdot x^3 + \ldots - p_{l-1} \cdot x^{l-1} + p_l \cdot x^l \approx (y)^{-\frac{1}{K}}$$

The approximating polynomial $p_A(x)$ approximates the $K$-th root family function $(y)^{\pm\frac{1}{K}}$, where $y=c_0+x$ for some center of expansion $c_0$. Additionally, the approximating polynomial $p_A(x)$ may include a polynomial of any degree as indicated by the value $l$, to approximate the $K$-th root family function $(y)^{\pm\frac{1}{K}}$.

**[0018]** Approximating polynomial coefficients stored in an instruction may be referred to as instruction-embedded polynomial coefficients. As described in greater detail below in connection with Figs. 3 and 4, instruction-embedded polynomial

coefficients may enable a processor system to determine a runtime approximating polynomial of a $K$-th root family function $(y)^{\pm\frac{1}{K}}$ using only instruction memory accesses. Furthermore, the processor system may use only instruction memory accesses to determine a $K$-th root family function approximation value based on the approximating polynomial. Although the apparatus and methods described herein relate generally to $K$-th root family functions of the form $(y)^{\pm\frac{1}{K}}$, instruction-embedded polynomial coefficients may be used to determine any runtime polynomial and runtime polynomial value that approximate any transcendental or algebraic function.

**[0019]** Fig. 1 is a flow diagram illustrating an example method for determining and storing approximating polynomial coefficient values. An approximating polynomial of a $K$-th root family function of the form $(y)^{\pm\frac{1}{K}}$ is determined (block 110) and coefficients of the approximating polynomial are rounded to eight significant bits (block 120) and embedded or otherwise stored in an instruction (block 130). The resulting instruction may be stored in an instruction memory (not shown). The approximating polynomial determined at block 110 may include any number of terms or term coefficients and, thus, may be a second-degree polynomial, a third-degree polynomial, a fourth-degree polynomial, etc. However, the example method for determining and storing approximating polynomial coefficient values is based on a third-degree approximating polynomial.

**[0020]** A $K$-th root family function approximation value may be determined for any input variable value $y$ within the range $1 \le y < 2$. The input variable value $y$ may be represented in several forms, all of which may include a polynomial variable value $x$. For purposes of clarity, the input variable value $y$ is represented in two forms

below. A first form used to determine an approximating polynomial for an inverse function $(y)^{-1}$ (i.e., $K = 1$), may be written as $y = 1.5 + x$, where $-0.5 \le x < 0.5$. A second form of the input variable value $y$, which may be used to determine an approximating polynomial for an inverse square-root function $(y)^{-\frac{1}{2}}$, may be written as $y = 1 + x$, where the polynomial variable value $x$ represents a fractional or decimal portion of the input variable value $y$. For example, for a value of $y$ equal to 1.3, the input variable value $y$ may be written as $y = 1 + x$, where solving for $x$ yields $x = 0.3$.

**[0021]** Generally, an approximating polynomial $p_a(x)$ of a $K$-th root family function $(y)^{\pm\frac{1}{K}}$ may be determined using a minimax approximation. Alternatively, a Taylor series expansion or Chebyshev expansion could be used. A $K$-th root family function $(y)^{\pm\frac{1}{K}}$ is shown in Equation 2 in terms of the polynomial variable $x$. Furthermore, as shown in Equation 3 below, the approximating polynomial $p_a(x)$ may include coefficient values $a_0$ through $a_3$.

Equation 2 $$\left(\frac{1}{y}\right)^{\pm\frac{1}{K}} = \left(\frac{1}{1.5+x}\right)^{\pm\frac{1}{K}} or \left(\frac{1}{1+x}\right)^{\pm\frac{1}{k}}$$

Equation 3 $$p_a(x) = a_0 - a_1 \cdot x + a_2 \cdot x^2 - a_3 \cdot x^3$$

**[0022]** In Equation 3, the coefficient values $a_0$ through $a_3$ are used to determine 8-bit approximating polynomial coefficient values. In particular, the coefficient values $a_0$ through $a_3$ are respectively associated with a zeroth-degree term coefficient value $p_0$, a first-degree term coefficient value $p_1$, a second-degree term coefficient value $p_2$ and a third-degree term coefficient value $p_3$. Furthermore, the rounding operation (block 120) performed on the coefficient values $a_0$ through $a_3$ results in two 8-bit

values that include the respective coefficient values $p_0$ through $p_3$. Additionally, as shown in Equation 4 below, an approximating polynomial *p(x)* associated with the approximating polynomial $p_a(x)$ may include the coefficient values $p_0$ through $p_3$.

Equation 4 $$p(x) = p_0 - p_1 \cdot x + p_2 \cdot x^2 - p_3 \cdot x^3$$

The values or absolute values of the coefficient values $p_0$ through $p_3$ of Equation 4 may be stored in at least one instruction (block 130) during the compilation phase.

**[0023]** As can be seen in Fig. 1, the rounding operation (block 120) rounds the coefficient values $a_0$ through $a_3$ simultaneously. Such a simultaneous rounding operation may reduce the accuracy with which an approximating polynomial approximates the *K*-th root family function $(y)^{\pm\frac{1}{K}}$. Another method described in connection with Fig. 2 below may be used to determine the coefficient values $p_0$ through $p_3$ to more accurately determine an approximating polynomial.

**[0024]** Fig. 2 is a flow diagram illustrating another example method for determining approximating polynomial coefficient values. The example method described in connection with Fig. 2 may provide a more accurate approximating polynomial of the *K*-th root family function $(y)^{\pm\frac{1}{K}}$. In particular, in contrast to the example method of Fig. 1, the example method shown in Fig. 2 uses independent rounding operations for the coefficient values $a_0$ through $a_3$, which results in a more accurate representation of the approximating polynomial.

**[0025]** More specifically, after rounding the coefficient values $a_0$ and $a_1$, a second approximating polynomial, which includes a second coefficient value, is determined. After rounding the second coefficient value, a third approximating polynomial that includes a third coefficient value is determined. In this manner, the example method

of Fig. 2 ensures greater approximation accuracy when determining an approximating polynomial because each successive coefficient value is based on a previously fixed coefficient value.

**[0026]** Now turning in detail to Fig. 2, a first approximating polynomial to a *K*-th root family function $(y)^{\pm\frac{1}{K}}$ is determined (block 210) and is similar to the approximating polynomial $p_a(x)$ of Equation 3 above. The first approximating polynomial includes coefficients $a_0$ and $a_1$. The zeroth-degree term coefficient $p_0$ and the first-degree term coefficient $p_1$ are determined by rounding the coefficients $a_0$ and $a_1$ at block 220 to 8-bit values. The coefficient $p_1$ may be used at block 230 to determine a second approximating polynomial.

**[0027]** As shown in Equation 5, the first-degree term coefficient $p_1$ may be multiplied by the polynomial variable value $x$, resulting in a product that is subtracted from the inverse square root function of the input variable value $y$. A second approximating polynomial shown in Equation 6 approximates the function of Equation 5 and is determined at block 230.

Equation 5 $$\frac{1}{\sqrt{y}} - p_1 \cdot x$$

Equation 6 $$1 + b_2 \cdot x^2 + b_3 \cdot x^3$$

**[0028]** As shown in Equation 6, the second approximating polynomial includes a coefficient value $b_2$. A second-degree term coefficient value $p'_2$ is determined by rounding the coefficient value $b_2$ to an 8-bit value (block 240).

**[0029]** The second-degree term coefficient $p'_2$ may be multiplied twice by the polynomial variable value $x$, resulting in a product that is subtracted from Equation 5

to produce a function according to Equation 7 below. A third approximating polynomial shown in Equation 8, which approximates the function of Equation 7, is then determined (block 250).

Equation 7 $$\frac{1}{\sqrt{y}} - p_1 \cdot x - p_2 \cdot x^2$$

Equation 8 $$1 + g_3 \cdot x^3$$

**[0030]** As shown in Equation 8, the third approximating polynomial includes a coefficient value $g_3$. A third-degree term coefficient value $p'_3$ is determined by rounding the coefficient value $g_2$ to an 8-bit value (block 260).

**[0031]** Equation 9 below shows an approximating polynomial of the *K*-th root family function $(y)^{\pm\frac{1}{K}}$ including the coefficient values $p_0$ through $p'_3$.

Equation 9 $$p(x) = p_0 - p_1 \cdot x + p'_2 \cdot x^2 - p'_3 \cdot x^3$$

The values or absolute values of the coefficient values $p_0$ through $p'_3$ of Equation 9 may be stored in at least one instruction (block 270) during a compilation. Additionally, the coefficient values $p_0, p_1, p_2$ and $p_3$ described in connection with Fig. 1 and the coefficient values $p_0, p_1, p'_2$ and $p'_3$ described in connection with Fig. 2 may be calculated once prior to a compilation phase and used multiple times during a runtime phase to determine a runtime polynomial value. The runtime polynomial value may be associated with a runtime approximating polynomial value of a *K*-th root family function $(y)^{\pm\frac{1}{K}}$ as set forth in greater detail below.

**[0032]** In the following description, the coefficient values $p_0$, $p_1$, $p_2$ and $p_3$ and the coefficient values $p_0$, $p_1$, $p'_2$ and $p'_3$ are referred to as the coefficient values $p_0$, $p_1$, $p_2$ and $p_3$.

**[0033]** The methods for determining a runtime approximating polynomial value of a $K$-th root family function $(y)^{\pm\frac{1}{K}}$ described below may be implemented on an integer-based processor system as well as a non-integer based processor system (e.g., a floating-point processor system). However, in the case of an integer-based processor system implementation, it may be useful to scale certain values such as, for example, the approximating polynomial coefficient values $p_0$ through $p_3$ to prevent loss of accuracy, resolution or overflow of subsequently calculated values. For example, if a 32-bit value is to be multiplied by a 10-bit value using a 32-bit operation, it may be useful to first scale the 32-bit value down to a 22-bit value to prevent overflow during the 32-bit multiplication operation.

**[0034]** In addition to scaling, it may also be useful to represent decimal or fractional values as integers when using an integer-based processor system. In particular, the methods described in connection with Figs. 3 and 4 use a Qk notation to represent decimal or fractional values as whole number integers, where the least significant bit of a value is related to $2^{-k}$.

**[0035]** In general, the example methods described in connection with Figs. 3 and 4 may be implemented using any integer-based or non-integer-based processor system capable of operations of any bit-length (e.g., 32-bit operations, 64-bit operation, etc.). However, for purposes of clarity, the example methods of Figs. 3 and 4 are described in connection with a 32-bit integer-based processor system. Thus, scaling methods and Qk notation used in connection with the examples of Figs. 3 and 4 are based on a maximum bit-length of 32 bits.

**[0036]** Fig. 3 is a flow diagram illustrating an example method for determining a runtime approximating polynomial value of an inverse function $(y)^{-1}$ (i.e., $K$=1) using instruction-embedded polynomial coefficient values. The example method of Fig. 3 includes four instruction-embedded polynomial coefficient values that are generally referred to as a zeroth-degree term coefficient value $p_0$, a first-degree term coefficient value $p_1$, a second-degree term coefficient value $p_2$ and a third-degree term coefficient value $p_3$.

**[0037]** During a runtime phase, a processor system (such as that shown in Fig. 6) may perform the example method depicted in Fig. 3 to determine a runtime approximating polynomial of an inverse function $(y)^{-1}$. By performing the operations of blocks 305-350 during a runtime phase, a runtime approximating polynomial may be used to determine a runtime approximating polynomial value of an inverse function $(y)^{-1}$. Specifically, the operations performed at blocks 305-350 reconstruct a runtime approximating polynomial similar to the approximating polynomial $p(x)$ of Equation 4 using the instruction-embedded polynomial coefficient values $p_0$, $p_1$, $p_2$, and $p_3$, the input variable value $y$, the polynomial variable value $x$ and a series of computational operations.

**[0038]** At runtime, the input variable value $y$ may be provided in Q31 format and, as described in connection with Fig. 1, may be represented as $y = 1.5 + x$. The polynomial variable value $x$ may be extracted from the input variable value $y$ and formatted (block 305) through a series of operations. Performing a 1-bit logical shift left on the input variable value $y$ results in a value $y$-1 in Q32 format. A value of 0.5 is then subtracted from the value y-1 to produce y-1.5, resulting in the polynomial variable value $x$ (i.e., $x$=$y$-1.5) in Q32 format. A 22-bit arithmetic shift right formats the polynomial variable value $x$ to Q10 format.

**[0039]** The third-degree term coefficient value $p_3$ may be retrieved from instruction memory and multiplied by the polynomial variable value $x$ (block 310), where $p_3$ and $x$ may each be represented in Q10 format. Multiplying the third-degree term coefficient value $p_3$ by the polynomial variable value $x$ results in a product value $p_3 \cdot x$ in Q20 format.

**[0040]** A first-degree polynomial is then determined (block 320) by fetching or retrieving the second-degree term coefficient value $p_2$ from instruction memory, scaling it to Q20 format and subtracting the product value $p_3 \cdot x$ from the second-degree term coefficient value $p_2$ as shown in Equation 10 below.

Equation 10 $$p_2 - p_3 \cdot x$$

As described below, the first-degree polynomial determined at block 320 may then be used to determine a second-degree polynomial.

**[0041]** A second-degree polynomial is determined (block 340) by retrieving the first-degree term coefficient value $p_1$ from instruction memory, formatting $p_1$ to Q16 format, multiplying the polynomial variable value $x$, which is in Q10 format, by a first-degree polynomial (e.g., the first-degree polynomial shown in Equation 10) and subtracting the result to the first-degree term coefficient value $p_1$. The second-degree polynomial is in Q30 format and may be represented as shown in Equation 11 below.

Equation 11 $$p_1 - p_2 \cdot x + p_3 \cdot x^2$$

**[0042]** A runtime approximating polynomial of the inverse function is then determined (block 350) by retrieving the zeroth-degree term coefficient value $p_0$ from

instruction memory, formatting $p_0$ to Q14 format, multiplying the polynomial variable value $x$ by a second-degree polynomial (e.g., the second-degree polynomial shown in Equation 11) and subtracting the result from the zeroth-degree term coefficient value $p_0$. The subtraction operation results in a runtime approximating polynomial value $p_v(x)$ of an inverse function in Q14 format and may be evaluated according to Equation 12 below.

Equation 12 $$u' = p_v(x) = p_0 - p_1 \cdot x + p_2 \cdot x^2 - p_3 \cdot x^3 \approx \frac{1}{1.5 + x}$$

The inverse function $(y)^{-1}$ is shown as $\frac{1}{1.5+x}$ and is approximated by a runtime approximating polynomial $p_v(x)$. The runtime approximating polynomial $p_v(x)$ may be used to determine an intermediate inverse function approximation value $u'$.

**[0043]** In general, if an application is configured to determine a more precise approximation (i.e., more significant bits) of the inverse function (block 351), a self-correcting process may be performed at block 352 on the intermediate inverse function approximation value $u'$ to determine an inverse function approximation value $u$ having a greater number of significant bits. For example, the intermediate inverse function approximation value $u'$ may be represented by an 8-bit value, while the inverse function approximation value $u$ may be represented by a more precise 16-bit value. If an application is not configured to determine a more precise value (block 351), then the inverse function approximation value $u$ is set equal to the intermediate inverse function approximation value $u'$.

**[0044]** Fig. 4 is a flow diagram illustrating an example method for determining a runtime approximating polynomial value of an inverse square root function and a

square root function using instruction-embedded polynomial coefficient values. The instruction-embedded polynomial coefficient values used in this example method generally include the zeroth-degree term coefficient value $p_0$, the first-degree term coefficient value $p_1$ and the second-degree term coefficient value $p_2$.

**[0045]** During a runtime phase, a processor system (such as that shown in Fig. 6) may perform the example method depicted in Fig. 4 to determine a runtime approximating polynomial of an inverse square root function. A runtime approximating polynomial may be used to determine a runtime approximating polynomial value of an inverse square root function and a square root function, which may be respectively associated with an inverse-square root approximation value and a square root approximation value. An inverse square root approximation value and/or a square root approximation value may be determined during a runtime phase by performing the operations of blocks 405-460. Specifically, the operations performed at blocks 405-460 reconstruct a runtime approximating polynomial similar to the approximating polynomial $p(x)$ of Equation 3 at a runtime phase using the instruction-embedded polynomial coefficient values $p_0$, $p_1$ and $p_2$, the input variable value $y$, the polynomial variable value $x$ and a series of computational operations.

**[0046]** At runtime, the input variable value $y$ may be given as an input value in Q31 format and, as described in connection with Fig. 1, may be represented as $y = 1 + x$. The polynomial variable value $x$ represents the decimal or fractional portion, which may be extracted from the input variable value $y$. Isolating the decimal or fractional portion includes performing a 1-bit logical shift left (block 405) on the input variable value $y$, resulting in the polynomial variable value $x$.

**[0047]** The second-degree term coefficient value $p_2$ may be retrieved from instruction memory and multiplied by the polynomial variable value $x$ (block 410),

where $p_2$ and $x$ may each be represented in Q10 format. Multiplying the second-degree term coefficient value $p_2$ and the polynomial variable value $x$ results in a product value $p_2 \cdot x$ in Q20 format, where the second-degree term coefficient value $p_2$ is associated with a runtime invariant value stored in instruction memory and the polynomial variable value $x$ is provided at runtime (i.e., is a runtime variant value).

**[0048]** A first-degree polynomial is then determined (block 420) by fetching or retrieving the first-degree term coefficient value $p_1$ from instruction memory and scaling it to Q20 format and subtracting the product value $p_2 \cdot x$ from the first-degree term coefficient value $p_1$ as shown in Equation 13 below.

Equation 13 $$p_1 - p_2 \cdot x$$

As shown in Equation 13, the first-degree polynomial determined at block 420 includes the polynomial variable value $x$ and the approximating polynomial coefficient values $p_1$ and $p_2$. As described below, the first-degree polynomial determined at block 420 may then be used to determine a second-degree polynomial.

**[0049]** As depicted by the example method in Fig. 4, a second-degree polynomial is determined (block 440) by multiplying the polynomial variable value $x$, which is in Q10 format, by a first-degree polynomial (e.g., the first-degree polynomial shown in Equation 13). Furthermore, as depicted by Equation 14 below, the second-degree polynomial includes a second-degree term having the second-degree term coefficient value $p_2$ and a first-degree term having the first-degree term coefficient value $p_1$.

Equation 14 $$p_1 \cdot x - p_2 \cdot x^2$$

The second-degree polynomial shown in Equation 14 may be represented in Q30 format and may be used to determine a runtime approximating polynomial of the inverse square root function.

**[0050]** A runtime approximating polynomial of the inverse square root function is determined by retrieving the zeroth-degree term coefficient value $p_0$ from instruction memory, formatting $p_0$ to Q30 format and subtracting a second-degree polynomial (e.g., the second-degree polynomial shown in Equation 14) from the zeroth-degree term coefficient value $p_0$ (block 440). The subtraction operation results in a runtime approximating polynomial value $p_v(x)$ in Q30 format that is associated with a runtime approximating polynomial of an inverse square root function.

**[0051]** A runtime approximating polynomial may be used to calculate an intermediate inverse square root approximation value $v'$ based on the approximating polynomial coefficient values $p_0$, $p_1$ and $p_2$ and the polynomial variable value $x$. The intermediate inverse square root approximation value $v'$ is determined (block 450) by performing a rounding operation on the runtime approximating polynomial value $p_v(x)$. More specifically, the rounding operation may be used to convert the runtime approximating polynomial value $p_v(x)$ in Q30 format to a runtime approximating polynomial value $p_v(x)$ in Q8 format by adding a binary one to the twenty-first bit position of the runtime approximating polynomial value $p_v(x)$ and performing a 22-bit logical shift right operation. The runtime approximating polynomial value $p_v(x)$ in Q8 format includes the intermediate inverse square root approximation value $v'$ as depicted in Equation 15 below.

Equation 15 $$v' = p_v(x) = p_0 - p_1 \cdot x + p_2 \cdot x^2 \approx \frac{1}{\sqrt{1+x}}$$

The inverse square root function of the input variable value $y$ is shown as $\frac{1}{\sqrt{1+x}}$ and is approximated by a runtime approximating polynomial that is used to determine the inverse square root approximation value $v'$.

**[0052]** In general, if an application is configured to determine a more precise approximation (i.e., more significant bits) of the inverse square root function (block 451), a self-correcting process may be performed at block 452 on the intermediate inverse square root approximation value $v'$. Thus, the self-correcting process (block 452) determines the inverse square root approximation value $v$ based on the intermediate inverse square root approximation value $v'$. If an application is not configured to determine a more precise value (block 451), then the inverse square root approximation value $v$ is set equal to the intermediate inverse square root approximation value $v'$ from block 450 and control is passed to block 455 where an application may choose to determine a square root approximation value $w$.

**[0053]** If an application is not configured to determine a square root approximation value $w$ (block 455), then the process may end with the inverse square root approximation value $v$ as a result. On the other hand, if an application is configured to determine the square root approximation value $w$, then the inverse square root approximation value $v$ is multiplied by the input variable value $y$ (block 460) as shown in Equation 16 below.

Equation 16
$$w = y \cdot v \approx (1+x) \cdot \frac{1}{\sqrt{1+x}} = \sqrt{1+x}$$

where, $$\frac{1}{\sqrt{1+x}} \approx p_0 - p_1 \cdot x + p_2 \cdot x^2$$

As shown in Equation 16, the square root approximation value $w$ approximates the square root function of the input variable value $y$ (i.e., $(y)^{\frac{1}{2}}$).

**[0054]** Although the approximation values $v$ and $w$ are depicted as being calculated using 8-bit coefficient values, these values may be calculated using larger bit length values if desired. For example, if the runtime invariant approximating coefficient values $p_1$ and $p_2$ are stored in instruction memory or retrieved from instruction memory as 16-bit values, a 16-bit value may be calculated at block 450 that includes the intermediate inverse square root approximation value $v'$.

**[0055]** One example method that may be used for retrieving 16-bit coefficient values from memory includes separating a 16-bit coefficient into two 8-bit values and storing each of the 8-bit values in a different instruction during a compilation phase. The instructions may be sequenced so that during a runtime phase, each 8-bit value that is stored in a different instruction may be easily concatenated to form a 16-bit coefficient. This method for retrieving coefficients having more than eight bits from instruction memory during runtime may be used for any number of coefficients having any desired bit length. Coefficients having more than eight bits may be implemented by using a processor system that supports having larger bit-length values stored in instructions.

**[0056]** Fig. 5 is a flow diagram that depicts an example method for performing a self-correcting process that may be used to determine a function approximation value based on an intermediate function approximation value. In general, the self-correcting process may be used to determine a function approximation value $f$ (i.e., a $K$-th root family function approximation value of the $K$-th root family function $(y)^{\pm\frac{1}{K}}$) that includes a more precise representation of the intermediate function approximation value $f'$. For example, the intermediate function approximation value $f'$ may be an

8-bit value. However, by performing the self-correcting process on the intermediate function approximation value $f'$, a more precise value may be determined, such as, for example, a 16-bit value that includes the function approximation value $f$. The intermediate function approximation value $f'$ is associated with the intermediate approximation values $u'$ and $v'$ of Figs. 3 and 4. For example, if an application is configured to determine the inverse square root approximation value $v$, then the intermediate function approximation value $f'$ is set equal to the intermediate inverse square root function approximation value $v'$ and the resulting function approximation value $f$ includes the inverse square root approximation value $v$.

**[0057]** The self-correcting process shown in Fig. 5 may be used to determine the function approximation value $f$ based on the intermediate function approximation value $f'$ and the input variable value $y$. For purposes of clarity, the intermediate function approximation value $f'$ is depicted as being based on the intermediate inverse square root approximation $v'$. However, the self-correcting process may also be performed on the intermediate inverse function $u'$ described in connection with Fig. 3 or any $K$-th root family function $(y)^{\pm\frac{1}{K}}$.

**[0058]** The intermediate function approximation value $f'$ may be mathematically represented in terms of an inverse square root function of the input variable value $y$ as set forth in Equation 17 below. Alternatively, the intermediate function approximation value $f'$ may be more precisely represented in terms of the inverse square root function of the input variable value $y$ and an error approximation value $e$ as set forth in Equation 18 below.

Equation 17 $$f' \approx \frac{1}{\sqrt{y}}$$

Equation 18 $$f' = \frac{1}{\sqrt{y}} \cdot (1 + e)$$

**[0059]** As shown in Equation 17, the intermediate function approximation value $f'$ is approximately equal to the inverse square root function of the input variable value $y$. Alternatively, Equation 18 shows that the intermediate approximation value $f'$ may be equal to the inverse square root function of the input variable value $y$ multiplied by a quantity $1+e$. The error approximation value $e$ is associated with an approximation factor introduced by determining the intermediate approximation value $f'$ using an approximating polynomial value (e.g., the approximating polynomial value $p_v(x)$ of Equation 15). Persons of ordinary skill in the art will readily appreciate that the self-correcting process may be used to reduce the effect of the error approximation value $e$ on the function approximation value $f$.

**[0060]** As depicted in Fig. 5, the intermediate function approximation value $f'$ is raised to the power of the exponent scaling value $K$ (block 510). The value of $K$ is equal to two in the case of the intermediate inverse square root approximation value $v'$. Thus, the operation at block 510 determines a scaled intermediate function approximation value $f'^2$, which is alternatively shown in Equation 19 below. The scaled intermediate function approximation value $f'^2$ is multiplied by the input variable value $y$ (block 520) to determine a product value $f'^2 \cdot y$ as shown in Equation 20 below.

Equation 19 $$\left(\frac{1}{\sqrt{y}}\cdot(1+e)\right)^{K}=\frac{1}{y}\cdot(1+e)^{2}$$

Equation 20 $$y\cdot\frac{1}{y}\cdot(1+e)^{2}=(1+e)^{2}$$

Because the intermediate function approximation value $f'$ is in Q9 format and the input variable value $y$ is in Q16 format, the multiplication operation of blocks 510 and 520 may result in an overflow when performed using a 32-bit processor system. The product value $f'^{2}\cdot y$, as shown in Equation 20, may be represented in Q32 format. Furthermore, the product value $f'^{2}\cdot y$, which is in Q32 format, may include a binary one in bit position 31 (i.e., the most significant bit of a 32-bit register) and may be interpreted as a signed value. Thus, due to the overflow at blocks 510 and 520, the product value $f'^{2}\cdot y$ approximates a value of one subtracted from Equation 20 as shown in Equation 21 below.

Equation 21 $$f'^{2}\cdot y\approx 2\cdot e+e^{2}$$

**[0061]** Next, an arithmetic shift operation (block 530) may be performed to format the product value $f'^{2}\cdot y$ to an appropriate bit-length for subsequent mathematical operations. An arithmetic shift operation is used to preserve the sign-bit of the Q32 format signed product value $f'^{2}\cdot y$. In particular, the arithmetic shift operation is performed as an 11-bit arithmetic shift right operation, which results in a product value $f'^{2}\cdot y$ in Q21 format.

**[0062]** The product value $f'^2 \cdot y$, which is in Q21 format, is multiplied by the intermediate function approximation value $f'$, which is in Q9 format, at block 540, resulting in a product value $f'^3 \cdot y$ in Q30 format. The product value $f'^3 \cdot y$ is then divided by the exponent scaling value $K$ (block 543). The value of $K$ is equal to two for the intermediate inverse square root approximation value $v'$. Thus, the operation at block 543 determines a scaled product value $\frac{f'^3 \cdot y}{2}$, which may be formatted in Q30 format. A 22-bit logical shift left operation is performed on the intermediate function approximation value $f'$ (block 545) after which the product value $\frac{f'^3 \cdot y}{2}$ in Q30 format is subtracted from the resulting intermediate function approximation value $f'$ (block 550). The subtraction operation at block 550 results in a 16-bit value in Q30 format that includes the function approximation value $f$. The function approximation value $f$ includes the inverse square root approximation value $v$. Additionally, as a result of the self-correcting process, the inverse square root approximation value $v$ is represented with greater precision (i.e., a 16-bit value) than the intermediate inverse square root approximation value $v'$ (i.e., an 8-bit value determined at blocks 405-450 of Fig. 4).

**[0063]** Although a 16-bit function approximation value $f$ may be determined using the methods described in connection with Fig. 5, a function approximation value $f$ having more significant bits (i.e., of greater precision) may be used instead. In particular, the function approximation value $f$ may be determined to a precision equivalent to the input variable value $y$ and/or the polynomial variable value $x$ provided in Figs. 3 and 4. For example, on a 64-bit processor system a 64-bit input

variable value $y$ may be used to enable the methods of Figs. 3, 4 and 5 to determine a 64-bit function approximation value $f$.

**[0064]** Additionally, multiple iterations of the self-correcting process described in connection with Fig. 5 may be performed to increase the precision of the function approximation value $f$. For example, for a 32-bit input variable value $y$, the methods of Figs. 3 and 4 may be used to determine an 8-bit intermediate function approximation value $f'$. However, a 32-bit function approximation value $f$ may be determined by performing two iterations of the self-correcting process on the 8-bit intermediate function approximation value $f'$. Each iteration of the self-correcting process increases the precision of the function approximation value $f$ by a factor of two.

**[0065]** Fig. 6 is a block diagram of an example processor system 610 that may be used to implement the apparatus and methods described herein. As shown in Fig. 6, the processor system 610 includes a processor 612 that is coupled to an interconnection bus or network 614. The processor 612 includes a register set or register space 616, which is depicted in Fig. 6 as being entirely on-chip, but which could alternatively be located entirely or partially off-chip and directly coupled to the processor 612 via dedicated electrical connections and/or via the interconnection network or bus 614. The processor 612 may be any suitable processor, processing unit or microprocessor such as, for example, a processor from the Intel X-Scale™ family, the Intel Pentium™ family, etc. In the example described in detail below, the processor 612 is a thirty-two bit Intel processor, which is commonly referred to as an IA-32 processor. Although not shown in Fig. 6, the system 610 may be a multi-processor system and, thus, may include one or more additional processors that are

identical or similar to the processor 612 and which are coupled to the interconnection bus or network 614.

**[0066]** The processor 612 of Fig. 6 is coupled to a chipset 618, which includes a memory controller 620 and an input/output (I/O) controller 622. As is well known, a chipset typically provides I/O and memory management functions as well as a plurality of general purpose and/or special purpose registers, timers, etc. that are accessible or used by one or more processors coupled to the chipset. The memory controller 620 performs functions that enable the processor 612 (or processors if there are multiple processors) to access a system memory 624, which may include any desired type of volatile memory such as, for example, static random access memory (SRAM), dynamic random access memory (DRAM), etc. The I/O controller 622 performs functions that enable the processor 612 to communicate with peripheral input/output (I/O) devices 626 and 628 via an I/O bus 630. The I/O devices 626 and 528 may be any desired type of I/O device such as, for example, a keyboard, a video display or monitor, a mouse, etc. While the memory controller 620 and the I/O controller 622 are depicted in Fig. 6 as separate functional blocks within the chipset 618, the functions performed by these blocks may be integrated within a single semiconductor circuit or may be implemented using two or more separate integrated circuits.

**[0067]** The methods described herein may be implemented using instructions stored on a computer readable medium that are executed by the processor 612. The computer readable medium may include any desired combination of solid state, magnetic and/or optical media implemented using any desired combination of mass storage devices (e.g., disk drive), removable storage devices (e.g., floppy disks,

memory cards or sticks, etc.) and/or integrated memory devices (e.g., random access memory, flash memory, etc.).

**[0068]** Although certain methods, apparatus and articles of manufacture have been described herein, the scope of coverage of this patent is not limited thereto. To the contrary, this patent covers all methods, apparatus and articles of manufacture fairly falling within the scope of the appended claims either literally or under the doctrine of equivalents